## 接口工具调研

本次接口工具调研，主要根据平时测试工作所涉及的内容考虑，从下面几个角度进行分析考虑：用例管理、执行，接口文档生成，请求参数加密处理，以及执行结果生成，自动化集成等方面。

### 一、Swapper简介

#### 1. Swagger介绍

Swagger是一个流行的API开发框架，Swagger允许用户使用Swagger编辑器描述OAS （OpenAPI Specification，OAS），并使用Swagger UI可视化并自动生成OAS中定义的API文档。

Swagger 对整个API的开发周期都提供了相应的解决方案，是一个规范和完整的框架。包括API设计、API开发、API文档、API测试、API治理等，Swagger几乎支持所有语言。

Swagger的主要开源工具包括：

**Swagger Editor**: 是用来编辑接口文档的小程序，非常简单易用。swagger是通过YAML来定义接口规范，可以通过YAML格式（标准JSON可以自动转换成YAML格式）的接口文档生成不同语言的服务端和客户端代码。提供了在线编辑器和本地部署编辑器两种方式。

**Swagger UI**:提供了一个可视化的UI页面展示API文档。接口的调用方、测试、项目经理等都可以在该页面中对相关接口进行查阅和做一些简单的接口请求。该项目支持在线导入描述文件和本地部署UI项目。

**Swagger Codegen**: 是一个开源的代码生成器，通过它可以生成服务端和客户端的代码。支持通过jar包，docker，node等方式在本地化执行生成。

#### Springfox Swagger介绍

**Springfox Swagger是**用Spring框架构建自动的 JSON API文档的工具。Spring 基于Swagger规范，可以将基于SpringMVC和SpringBoot框架的项目代码，自动生成JSON格式的API文档。允许用户通过Swagger注解自定义接口文档，通过SwaggerUI在线显示。

步骤如下：

（1）引用pom.xml，添加依赖文件。

   <dependency>

            <groupId>io.springfox</groupId>

            <artifactId>springfox-swagger2</artifactId>

            <version>2.x</version>

     </dependency>

     <dependency>

            <groupId>io.springfox</groupId>

            <artifactId>springfox-swagger-ui</artifactId>

            <version>2.x</version>

     </dependency>

1. 配置Springfox文件。

Spring和 Swagger 整合时，可以通过注解注入相关配置。通过这些配置可以指定在Spring启动时扫描哪些Controller层的文件夹，另外可以指定API文档页的标题和描述信息等内容。

@Configuration

@EnableSwagger2

public class SpringFoxConfig {

@Bean

public Docket apiDocket() {

return new Docket(DocumentationType.SWAGGER\_2)

.select()

.apis(RequestHandlerSelectors.any())

.paths(PathSelectors.any())

.build();

}

}

1. 编写测试APIController类。

一般在Controller层，将详尽的API接口输入输出在代码中通过注解进行相关描述。

1. 在线查看及测试API文档。

完成API文档的编写工作之后，正常启动spring，假如后台端口为8080，那么访问[http://localhost:8080/swagger-ui.html](http://127.0.0.1:8080/swagger-ui.html)，可以访问接口界面。通过该界面，不仅可以看到自动生成的所有API文档信息，还可以对任意接口进行在线测试，非常方便。

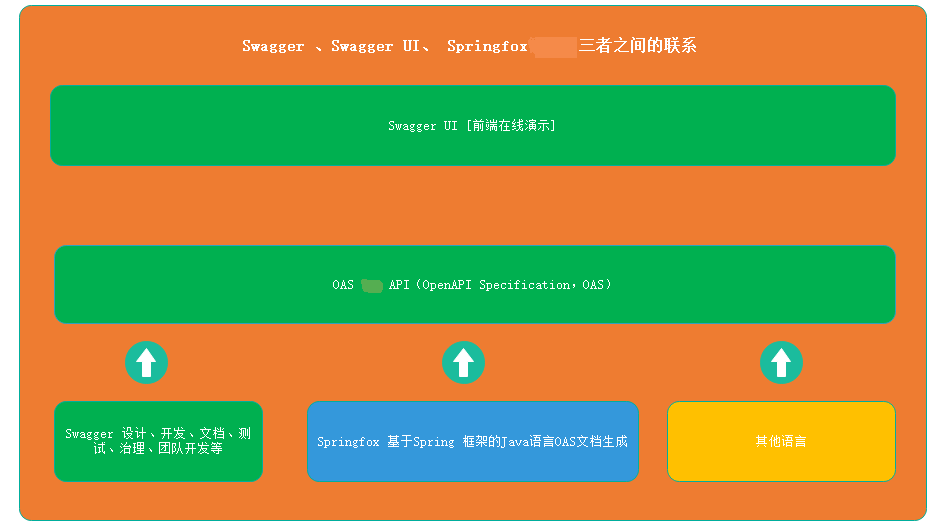
可参考以下内容：

<https://blog.csdn.net/amon1991/article/details/76774880>

<https://blog.csdn.net/dmw412724/article/details/80706326>

<https://www.vojtechruzicka.com/documenting-spring-boot-rest-api-swagger-springfox/>

Swagger、SwaggerUI、Springfox三者的联系见下图：



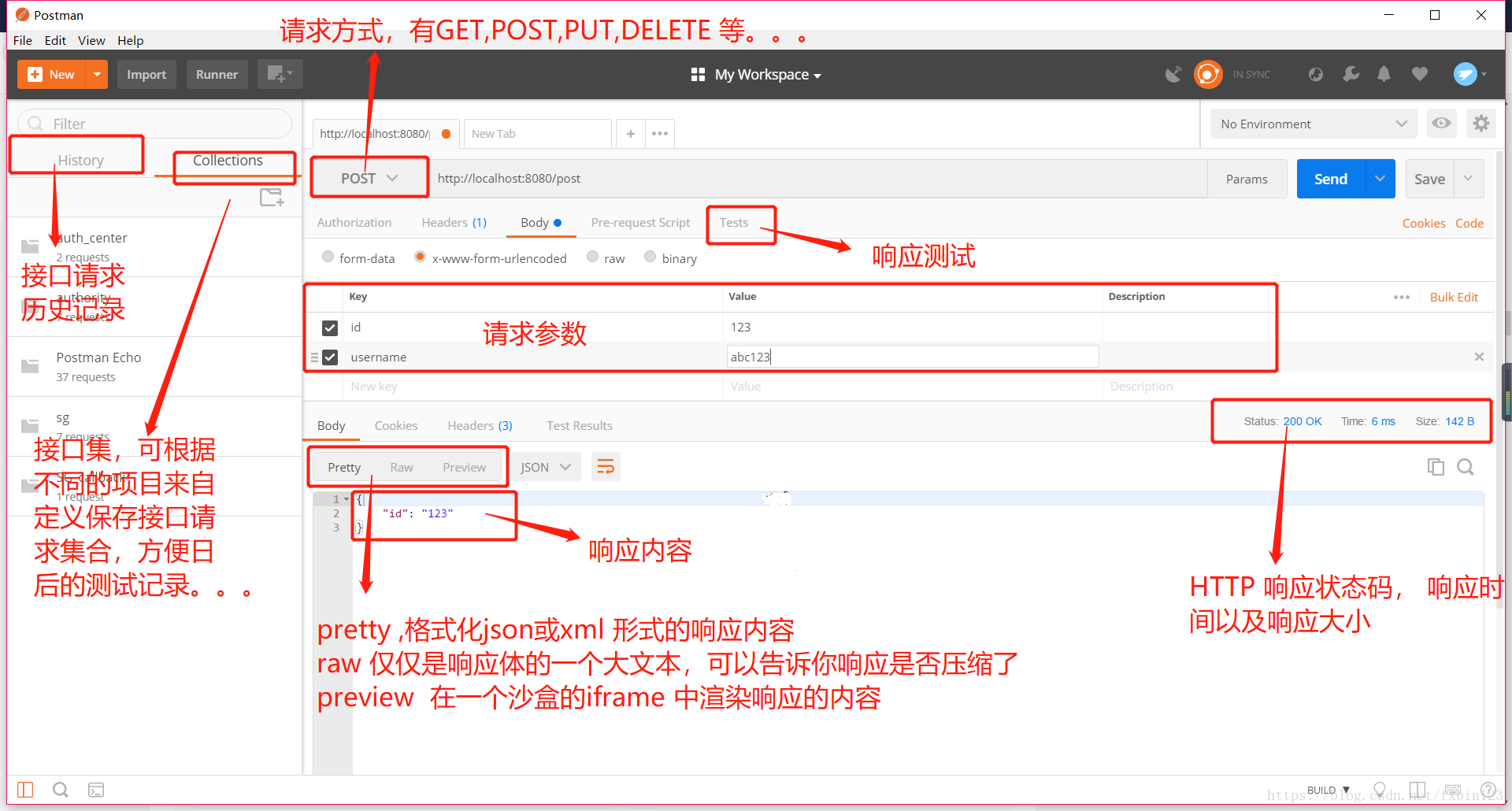
#### Swagger优缺点

优点：能实时同步API与文档。

缺点：功能复杂，对编码能力要求较高。

### 二、postman简介

Postman是一款功能强大的网页调试与发送网页HTTP请求的工具，可以发送get、 post或者其他方式的请求来调试接口，支持Windows、MAC、Linux操作系统。postman主界面见下图：



#### Postman的功能特性

###### 1.1 构建HTTP请求

HTTP请求中，构造的主要部分是url、Method、Headers、Body。Postman提供了友好的界面，可以让我们方便的构建这些内容。Postman发送的请求可以通过抓包工具直接获取。

get请求

post请求

post请求会发送请求实体，实体类型常用：x-www-from-urlencoded，application/JSON, text/plain，application/xml，multipart/form-data。

###### 1.2 参数化

Postman参数包括全局变量、环境变量、外部文件和局部变量。

全局变量：作用域是全局，比如我们要设置的token

环境变量：配置之后，让测试用例既能够在测试环境中执行，也能在正式环境中执行

局部变量：针对接口的个性化设置，如url等，作用域只限于当前接口范围内

外部文件变量：在执行测试集的时候可以设置循环次数，指定外部数据文件。外部文件可以是csv格式的，也可以是json格式的。变量的作用域优先级:

1. Global---全局

2. Enviroment --- 环境变量

3. Local --- 局部

4. Data ---- 数据

优先级从高到底

Data ---- > Local ---- > Enviroment ---- > Global

如果设置了一个变量，名为variable ，则引用该变量时按照如下形式：{{variable }}

###### 1.3 Authorization

Postman在验证权限机制方面提供了以下类型：

* Inherit auth from parent 在创建Collections时设置，文件夹下的请求都 使用父类的授权类型
* No Auth
* Bear Token
* Basic Auth
* Digest Auth
* OAuth 1.0
* OAuth 2.0
* Hawk Authentication
* AWS Signature
* NTML Authentication[Beta]

服务器需要对客户端进行身份验证: https://api.github.com/user

###### 1.4 Sandbox

Postman的Sandbox是一个JavaScript的执行环境，这使得你能够为请求写Pre-Request Scripts和Test Scripts。常用的库和工具见下表：

|  |  |
| --- | --- |
| Lodash | JS实用程序库 |
| jQuery |  |
| Tv4 JSON scheme validator | 根据json-schema草案的v4验证JSON对象 |
| CryptoJS | 支持的算法：AES，DES，EvpKDF，HMAC-MD5，HMAC-SHA1/3/256/512，MD5，PBKDF2，Rabbit，SHA1 / 3/224/256/512，TripleDES  不支持RSA算法，但是Postman支持eval函数，我们只要将RSA代码存入环境变量中，在需要的时候调用eval函数就可以解决 |
| Xml2Json |  |

###### 1.5 Pre-Request Script

Pre-Request Script是请求发送之前需要执行的代码片段，请求参数中需要随机数或者请时间戳，请求参数需要加密、参数化等都可以使用。

以关联参数为例,其中function(err,response)是回调函数。:

pm.sendRequest("https://postman-echo.com/get", function (err, response) {

var jsonObj = response.json();

pm.environment.set("addr", jsonObj.headers.host);

});

这个代码段默认只能发送get请求，不带首部的。发送post的代码需要自己编写：

var data = {"key1":"value1","key2":"value2"}

const echoPostRequest = {

url: 'https://postman-echo.com/post',

method: 'POST',

header: '**Content-Type:application/json**',

body: {

mode: 'raw',

raw: JSON.stringify(data)

}

};

pm.sendRequest(echoPostRequest, function (err, res) {

console.log(res.json());

});



Base64编码解码举例：

var CryptoJS = require("crypto-js");*//replace thie with script tag in browser env*

*//encrypt 写在pre-requestScript*

var rawStr = "hello world!";

var wordArray = CryptoJS.enc.Utf8.parse(rawStr);

var base64 = CryptoJS.enc.Base64.stringify(wordArray);

console.log('encrypted:', base64);

*//decrypt 写在TestScript里面*

var parsedWordArray = CryptoJS.enc.Base64.parse(base64);

var parsedStr = parsedWordArray.toString(CryptoJS.enc.Utf8);

**javascript中escape, encodeURI, encodeURIComponent的区别**

1. 对字符串(string)进行编码,其中 ASCII字母、数字、@\*/+ ，这几个字符不会被编码，其余的都会。
2. encodeURI方法不会对下列字符编码 ASCII字母、数字、~!@#$&\*()=:/,;?+'
3. encodeURIComponent方法不会对下列字符编码 ASCII字母、数字、~!\*()'

Postman脚本是基于JavaScript语言编写的，而导出的json格式的Postman脚本也无法直接在服务器运行，它需要在Newman中执行。所以要实现在Windows的cmd窗口或者linux系统中直接以命令的方式执行脚本，我们需要安装node.js，然后再在此基础上安装Newman。

请求时，如果需要带token认证，token放到请求首部中。

**发送带参数的get请求，和post请求类似：**

**var host = pm.environment.get("host");**

**const csrfTokenServlet = {**

**url: `https://${host}/{path}?csrfType=test`,**

**method: 'GET'**

**};**

**pm.sendRequest(csrfTokenServlet, (err, response) => {**

**if (err) {**

**console.log(err);**

**} else {**

**pm.environment.set("csrfToken", response.json()['data']);**

**console.log(response.json()['data'])**

**}**

**});**

**# content-type/x-www-form-urlencoded**

**再举3个post例子：**

**1.header首部是多个时，是个列表**

**var host = pm.environment.get("host");**

**var data = {**

**"organCode":"10000065",**

**}**

**const postRequest = {**

**url: `https://${host}/${path}`,**

**method: 'POST',**

**header:['Content-Type:application/x-www-form-urlencoded', 'Cookie:111111111'],**

**body: {**

**mode: 'x-www-form-urlencoded',**

**raw: "organCode=10000062"**

**}};**

**pm.sendRequest(postRequest, (err, response) => {**

**if (err) {**

**console.log(err);**

**} else {**

**console.log(response.json());**

**}**

**});**

**或者 表单格式**

**var host = pm.environment.get("host");**

**var data = {**

**"organCode":"10000065",**

**}**

**const postRequest = {**

**url: `https://${host}/${path}`,**

**method: 'POST',**

**header:'Content-Type:application/x-www-form-urlencoded',**

**body: {**

**mode: 'urlencoded',**

**urlencoded: "organCode=10000062"**

**}};**

**pm.sendRequest(postRequest, (err, response) => {**

**if (err) {**

**console.log(err);**

**} else {**

**console.log(response.json());**

**}**

**});**

**Json格式**

**# content-type/json**

**var data = {**

**"key1":"value1",**

**"key2":"value2",}**

**const echoPostRequest = {**

**url: 'http://ip:port/XXX,**

**method: 'POST',**

**header:'Content-Type:application/json',**

**body: {**

**mode: 'raw',**

**raw: JSON.stringify(data)**

**}};**

**pm.sendRequest(echoPostRequest, function (err, res) {**

**console.log(res.json());**

**});**

###### 1.6 Tests

接收到请求响应后，编写Tests-Script，制定响应断言、关联参数等。

postman可以在请求模块的Tests标签添加断言，比较方便的是，Postman已经提供了一些代码片段供我们使用。

例如，Status code :Code is 200判断状态码是否为200，会自动生成脚本

pm.test("Status code is 200", function () {

pm.response.to.have.status(200);

});

缓存需要的参数：

Var jsonData = JSON.parse(responseBody); responseBody是固定的响应。

Var data = jsonData.data;

Pm.environment.set(“data”,data);

###### 1.7 测试集管理

测试集Collections 和 文件夹Folder对API进行分类存放和管理。Collections比较类似文件夹，跟文件夹一样可以存放测试请求，但它不同于文件夹，通过测试集可以直接运行其下所有的请求，文件夹则不行。文件夹可以提供模块化管理。

Collections可以以json格式导入导出操作。

Postman生成格式化时间：

方法一：

var moment = require('moment');

var data = moment().format(" YYYYMMDDHHmmss");

console.log(data);

pm.globals.set("time", data);

方法二：

var myDate=new Date();

var year = myDate.getFullYear();

var month = myDate.getMonth() + 1;

var date = myDate.getDate();

var hours = myDate.getHours() + 1;

var min = myDate.getMinutes();

var seconds = myDate.getSeconds();

hours = hours < 10 ? "0" + hours : hours;

min = min < 10 ? "0" + min : min;

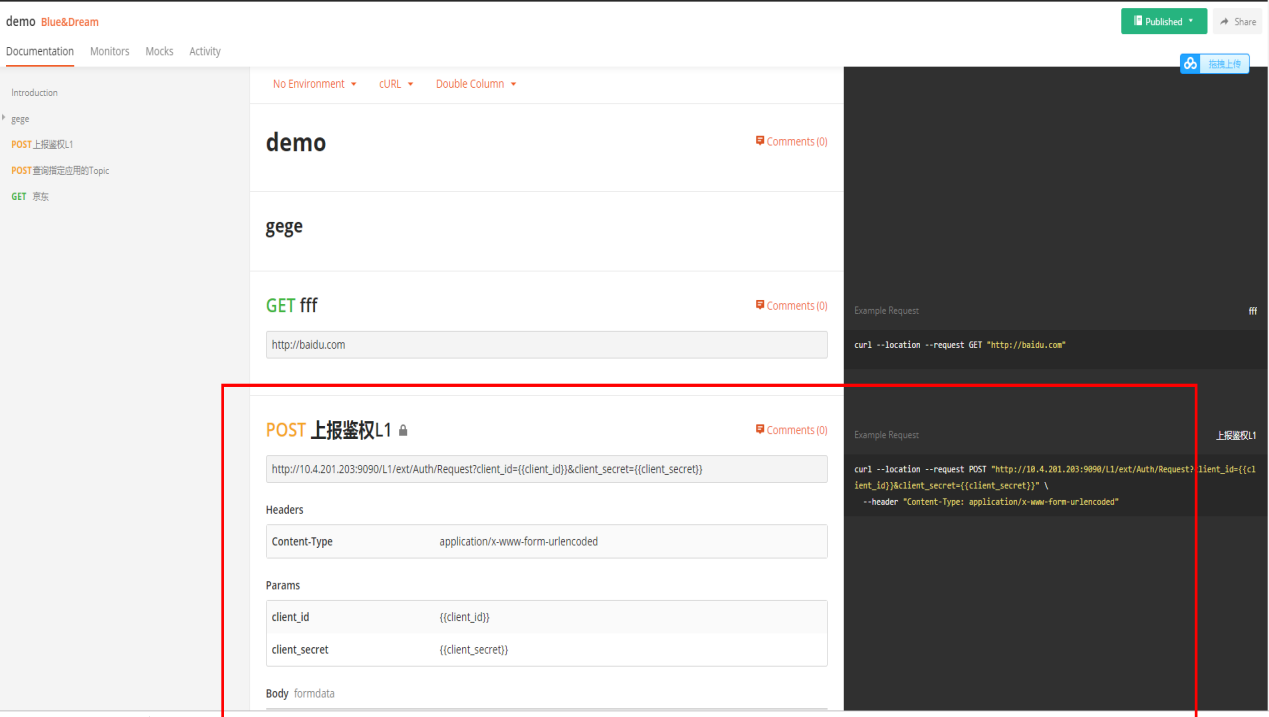
seconds = seconds < 10 ? "0" + seconds : seconds;

var date = year + "" + month + date + hours + min + seconds;

###### 1.8 接口文档及执行结果

Postman可以自动生成接口文档（见下图），生成的文档需要重新编辑。Postman执行结果是以json格式保存的。导出的json/url可以利用Newman进行自动化测试。

（在线接口文档<https://cloudy-desert-7760.postman.co/collections/4072071-4f5558bf-351f-462b-aa6c-6242e3b0e90a?workspace=c0225282-a327-48fd-8c67-31b41ed2ac4d>）



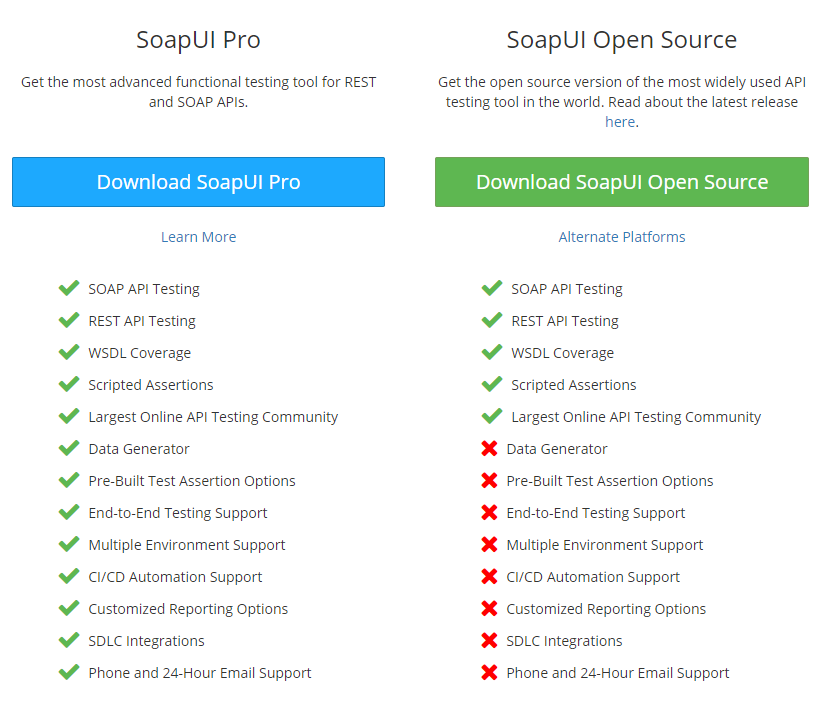
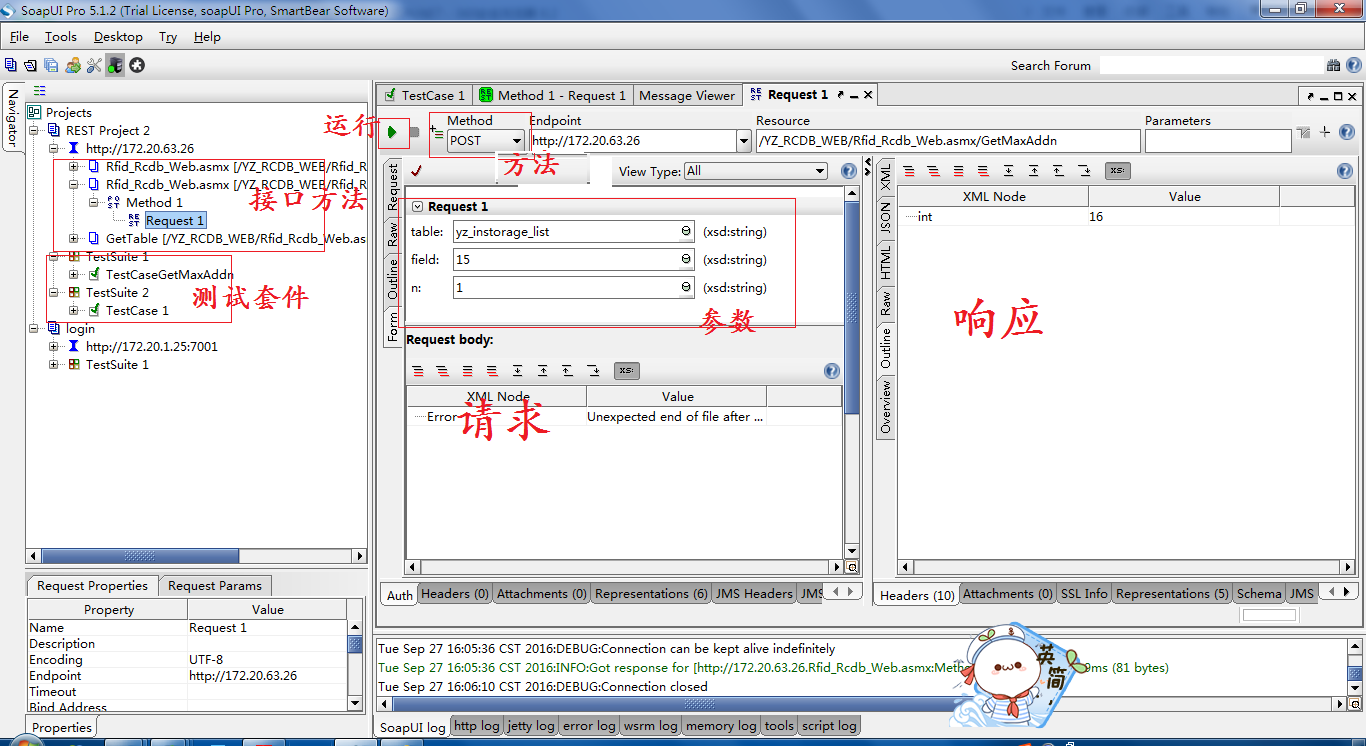
##### 2.postman优缺点

|  |  |
| --- | --- |
| **优点** | **缺点** |
| * 上手快，跨平台，免费版就已经非常强大，支持HTTP，HTTPS协议 * 脚本语言是JavaScript，自带各种代码模块 * 支持读取数据文件,json，csv，支持json Schema校验 * 支持抓包,保存历史记录,多终端同步用例 * 有命令行版本,Newman,方便放在服务器上运行,或者Jenkins做持续集成 * 如果项目有Swagger生成API文档，json文件可以直接导入Postman来生成用例 * 保存请求历史记录 | * 不支持：通过url搜索用例,通过命令行传参修改环境变量,不能指定执行失败的测试用例 * 不理想：不能操作文件相关的操作,不能读写数据库，不能使用非HTTP协议 * 不理想：封装公共函数,引入第三方，自己写的js库，用例间的跳转，重试等 * 断言不支持正则 * 执行测试集时，无法对Failed的用例单独执行，而且某个用例出现故障，线程就会停止 |

针对postman的功能特点，Postman比较适合做手工接口测试，适合在开发调试或者测试人员执行测试使用，在执行测试用例流程、可扩展性方面略显不足。Postman与AutoTest的使用类似。

### 三、soupUI简介

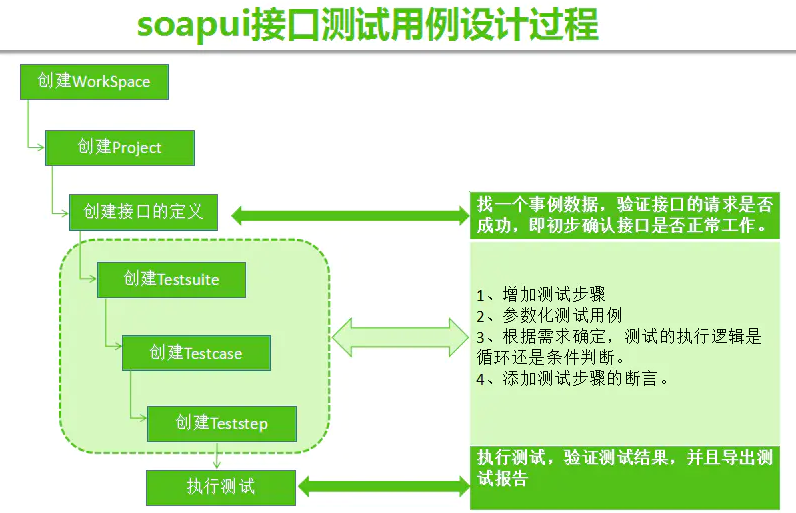
soapUI是一个开源的跨平台测试工具，通过soup/http来检查、调用、实现WebService的功能/负载/符合性测试。该工具既可作为一个单独的测试软件使用，也可利用[插件](http://baike.so.com/doc/1483288-1568456.html)集成到Eclipse，maven2.X，Netbeans 和intellij中使用。soapUI pro是soapUI的商业非开源版本，实现的功能较开源的soapUI更多（见下图）。

soupUI主界面介绍：  


#### soupUI的功能特性

|  |
| --- |
| 介绍soupUI之前，先介绍几个名词解释：  WSDL：指网络服务描述语言（Web Services Description Language），是一种使用XML编写的文档。这种文档可描述某个WebService，它可以规定服务的位置，如何访问服务，以及服务提供的操作。  Soup：指简单对象访问协议（Simple Object Access Protocol）是一种用于访问Web服务的协议。基于XML和HTTP，其通过XML来实现消息描述，然后通过HTTP实现消息传输。  Rest可以通过HTTP形式直接调用，利用HTTP的请求方式get/post等约定事务操作。 |

soupUI可用发送soup/http协议请求，根据界面操作很方便的构建请求。soupUI构建测试整体过程可以用下图表示：



###### 1.1构建请求

soupUI发送请求过程：新建WorkSpce — 新建Soap/Rest Project—创建TestSuite—创建TestCase—创建Step和请求—运行查看。

测试接口时，需先生成单接口用例，然后将其引入创建好的TestSuite->TestCase中的TestSteps。

发送请求通过抓包工具抓包时，需要设置Proxy。

###### 1.2参数化

soupUI中的参数值可通过DataSource引入，支持txt、Excel、JDBC等数据形式。也可以通过添加Properties和Proterty Transfer关联参数。

###### 1.3Authorization

soupUI在验证权限机制方面提供了以下类型：

* No Authorization
* Basic Auth
* NTML Auth
* SPNEGO/Kerberos
* OAuth 2.0

###### 1.4 Script

soupUI使用Groovy和JavaScript作为其脚本语言，但是默认使用Groovy语言。

Groovy是一种基于JVM的敏捷开发语言，它结合了Python、Ruby和Smalltalk的许多强大的特性，Groovy 代码能够与 Java 代码很好地结合，也能用于扩展现有代码。由于其运行在 JVM 上的特性，Groovy 可以使用Java 语言编写的库。

通过使用脚本可以对请求参数进行加解密处理、连接数据库和添加断言等一些扩展操作。

###### 1.5 断言

soupUI提供了多种方式断言。比如Contains、ReponseSLA（响应时间在指定时间内）、Xpath Match、Xquery Match、Groovy Script/JavaScript等。

###### 1.6接口文档及测试报告

soupUI不能自动生成接口文档。

soupUI执行测试用例时，可以指定执行的测试用例步骤，通过debugger调试。执行后生成XML和HTML格式的测试报告。

#### soupUI优缺点

|  |  |
| --- | --- |
| 优点 | 缺点 |
| * 调试接口方便，返回结果直观可以很容易看出，方便调试 * 功能强大，支持各种功能，还可以进行压力和安全测试 * 实现接口测试的操作步骤简单，有严谨的项目工程结构，方便管理项目 | * 接口多的情况下显示不直观，随着用例越多，层级结构复杂，如果有具体接口需要调试，操作繁琐 * 接口得一个一个添加到Testsuit中去，单接口做了修改后，需要同步到TestSuit中 * 界面控件过多，使用复杂 |

soupUI的优点胜于缺点，soupUI涵盖了postman的所有功能，除此之外，还可以进行负载/安全测试。soupUI在用例管理和自动化集成有一定的优势。

### 四、工具对比分析

|  |  |  |  |
| --- | --- | --- | --- |
| 工具 | 接口测试类型 | 自动化 | 脚本拓展能力 |
| Postman | API测试：HTTP协议  压力测试 | Postman+Newman + Jenkins 实现接口自动化测试 | JavaScript |
| SoupUI | API测试：Soap协议、HTTP协议  压力测试  安全测试 | SoapUI+Maven  + Jenkins 实现自动化测试 | Groovy脚本  JavaScript |